S/MIME WG
Internet Draft
Intended Status: Informational
Obsoletes: 3278 (once approved)
Expires: March 22, 2009

Sean Turner, IECA
Dan Brown, Certicom
September 22, 2008

# Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS) draft-ietf-smime-3278bis-02.txt

## Status of this Memo



## Copyright Notice



## Abstract

This document describes how to use Elliptic Curve Cryptography (ECC) public-key algorithms in the Cryptographic Message Syntax (CMS). The ECC algorithms support the creation of digital signatures and the exchange of keys to encrypt or authenticate content. The definition

of the algorithm processing is based on the NIST FIPS 186-3 for digital signature, NIST SP800-56A for key agreement, RFC 3565 for key wrap and content encryption, NIST FIPS 180-3 for message digest, and RFCs 2104 and 4231 for message authentication code standards.

Discussion

This draft is being discussed on the 'ietf-smime' mailing list. To subscribe, send a message to ietf-smime-request@imc.org with the single word subscribe in the body of the message. There is a Web site for the mailing list at <http://www.imc.org/ietf-smime/>.

Table of Contents

## 1. Introduction

The Cryptographic Message Syntax (CMS) is cryptographic algorithm independent. This specification defines a profile for the use of Elliptic Curve Cryptography (ECC) public key algorithms in the CMS. The ECC algorithms are incorporated into the following CMS content types:

- 'SignedData' to support ECC-based digital signature methods (ECDSA) to sign content
- 'EnvelopedData' to support ECC-based public-key agreement methods (ECDH and ECMQV) to generate pairwise key-encryption keys to encrypt content-encryption keys used for content encryption
- 'AuthenticatedData' to support ECC-based public-key agreement methods (ECMQV) to generate pairwise key-encryption keys to encrypt MAC keys used for content authentication and integrity.
- 'AuthEnvelopedData' to support ECC-based public-key agreement methods (ECMQV) to generate pairwise key-encryption keys to encrypt MAC keys used for authenticated encryption modes.

Certification of EC public keys is also described to provide public-key distribution in support of the specified techniques.

### 1.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [MUST].

### 1.2. Changes since RFC 3278

The following summarizes the changes:

- Abstract: The basis of the document was change to refer to NIST

FIPP 186-3 and SP800-56A.

- Section 1: A bullet was added to address AuthEnvelopedData.

- Section 2.1: A sentence was added to indicate [FIPS180-3] is used with ECDSA. Replaced reference to [X9.62] with [FIPS186-3].

- Section 2.1.1: The permitted digest algorithms were expanded from SHA-1 to SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512.





- Section 2.1.2 and 2.1.3: The bullet addressing integer "e" was deleted.

- Section 3: Added explanation of why static-static ECDH is not included.

- Section 3.1: The reference for DH was changed from CMS to CMS-ALG. Provided text to indicate fields of EnvelopedData are as in CMS.

- Section 3.1.1: The permitted digest algorithms for use with ECDH std and cofactor methods were expanded from SHA-1 to SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. Updated to include description of all KeyAgreeRecipientInfo fields. Parameters for id-ecPublicKey field changed from NULL to ABSENT or ECPoint.

- Section 3.2.1: The permitted digest algorithms for use with ECMQV were expanded from SHA-1 to SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. Updated to include description of all fields. Parameters for id-ecPublicKey field changed from NULL to ABSENT or ECPoint.

- Section 4.2: This section was added to address AuthEnvelopedData with ECMQV.

- Section 5: This section was moved to Section 8. The 1st paragraph was modified as the requirements are difficult to test. The requirements were updated for hash algorithms and recommendations for matching curves and hash algorithms. Also expanded to indicate which ECDH and ECMQV variants, key wrap algorithms, and content encryption algorithms are required for each of the content types used in this document.

- Section 6 (formerly 7): The S/MIME capabilities for ECDSA with SHA-224, SHA-256, SHA-384, and SHA-512 were added to the list of S/MIME Capabilities. Also updated to include S/MIME capabilities for ECDH and ECMQV using SHA2 algorithms as the KDF.

- Section 7.1 (formerly 8.1): Added sub-sections for digest, signature, originator public key, key agreement, content encryption, and message authentication code algorithms. SHA-224, SHA-256, SHA-384, and SHA-512 as well as SHA-224, SHA-256, SHA-384, and SHA-512 with ECDSA were added. Also added algorithm identifiers for ECDH std, ECDH cofactor, and ECMQV with SHA2 algorithms as the KDF. Message Authentication Code, Content Encryption, Key Wrap.





- Section 7.2 (formerly 8.2): Updated to include AuthEnvelopedData. Also, added text to address support requirement for compressed and uncompressed keys, changed pointers to ANSI X9.61 to PKIX (where ECDSA-Sig-Value is imported), changed pointers from SEC1 to NIST specs, and updated example of suppPubInfo to be AES-256. keyInfo's parameters changed from NULL to any associated parameters (AES wraps have absent parameters).

- Section 9: Replaced text, which was a summary paragraph, with an updated security considerations section. Paragraph referring to definitions of SHA-224, SHA-256, SHA-384, and SHA-512 is deleted.

- Added ASN.1 modules.

- Updated acknowledgements section.

## 2. SignedData using ECC

This section describes how to use ECC algorithms with the CMS SignedData format to sign data.

### 2.1. SignedData using ECDSA

This section describes how to use the Elliptic Curve Digital Signature Algorithm (ECDSA) with SignedData. ECDSA is specified in [FIPS186-3]. The method is the elliptic curve analog of the Digital Signature Algorithm (DSA) [FIPS186-3]. ECDSA is used with the Secure

Hash Algorithm (SHA) [FIPS180-3].

In an implementation that uses ECDSA with CMS SignedData, the following techniques and formats MUST be used.

### 2.1.1. Fields of the SignedData

When using ECDSA with SignedData, the fields of SignerInfo are as in [CMS], but with the following restrictions:

- digestAlgorithm MUST contain the algorithm identifier of the hash algorithm (see Section 7.1) which MUST be one of the following: id-sha1, id-sha224, id-sha256 identifies, id-sha384, and id-sha512.

- signatureAlgorithm contains the signature algorithm identifier (see Section 7.1): ecdsa-with-SHA1, ecdsa-with-SHA224, ecdsa-with-SHA256, ecdsa-with-SHA384, or ecdsa-with-SHA512.





- signature MUST contain the DER encoding (as an octet string) of a value of the ASN.1 type ECDSA-Sig-Value (see Section 7.2).

When using ECDSA, the SignedData certificates field MAY include the certificate(s) for the EC public key(s) used in the generation of the ECDSA signatures in SignedData. ECC certificates are discussed in Section 5.

### 2.1.2. Actions of the sending agent

When using ECDSA with SignedData, the sending agent uses the message digest calculation process and signature generation process for SignedData that are specified in [CMS]. To sign data, the sending agent uses the signature method specified in [FIPS186-3].

The sending agent encodes the resulting signature using the ECDSA-Sig-Value syntax (see Section 7.2) and places it in the SignerInfo.signature field.

### 2.1.3. Actions of the receiving agent

When using ECDSA with SignedData, the receiving agent uses the message digest calculation process and signature verification process

for SignedData that are specified in [CMS]. To verify SignedData, the receiving agent uses the signature verification method specified in [FIPS186-3].

In order to verify the signature, the receiving agent retrieves the integers r and s from the SignerInfo signature field of the received message.

## 3. EnvelopedData using ECC Algorithms

This section describes how to use ECC algorithms with the CMS EnvelopedData format.

### 3.1. EnvelopedData using (ephemeral-static) ECDH

This section describes how to use the ephemeral-static Elliptic Curve Diffie-Hellman (ECDH) key agreement algorithm with EnvelopedData, method C(1, 1, ECC CDH) from [SP800-56A]. Ephemeral-static ECDH is the elliptic curve analog of the ephemeral-static Diffie-Hellman key agreement algorithm specified jointly in the documents [CMS-ALG] and [CMS-DH].

In an implementation uses ECDH with CMS EnvelopedData, the following techniques and formats MUST be used.





The fields of EnvelopedData are as in [CMS], as ECDH is a key agreement algorithm the RecipientInfo kari choice is used. When using ECDH, the EnvelopedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key. ECC certificates are discussed in Section 5.

#### 3.1.1. Fields of KeyAgreeRecipientInfo

When using ephemeral-static ECDH with EnvelopedData, the fields of KeyAgreeRecipientInfo are as follows:

- version MUST be 3.

- originator MUST be the alternative originatorKey. The originatorKey algorithm field MUST contain the id-ecPublicKey object identifier (see Section 7.1). The parameters associated with id-ecPublicKey MUST be absent or ECPoint. NOTE: The previous version of this document required NULL be present,

support for this is OPTIONAL. The originatorKey publicKey field MUST contain the value of the ASN.1 type ECPoint (see Section 7.2), which represents the sending agent's ephemeral EC public key. The ECPoint in uncompressed form MUST be supported.

- ukm MAY be present or absent. However, message originators SHOULD include the ukm. As specified in RFC 3852 [CMS], implementations MUST support ukm message recipient processing, so interoperability is not a concern if the ukm is present or absent. When present, the ukm is used to ensure that a different key-encryption key is generated, even when the ephemeral private key is improperly used more than once, by using the ECC-Shared-Info as input to in the key derivation function (see Section 7.2).

- keyEncryptionAlgorithm MUST contain the key encryption algorithm object identifier (see Section 7.1). The parameters field contains KeyWrapAlgorithm. The KeyWrapAlgorithm is the algorithm identifier that indicates the symmetric encryption algorithm used to encrypt the content-encryption key (CEK) with the key-encryption key (KEK) and any associated parameters. Algorithm requirements are found in Section 8.

- recipientEncryptedKeys contains an identifier and an encrypted key for each recipient. The RecipientEncryptedKey KeyAgreeRecipientIdentifier MUST contain either the issuerAndSerialNumber identifying the recipient's certificate or the RecipientKeyIdentifier containing the subject key identifier from the recipient's certificate. In both cases, the





recipient's certificate contains the recipient's static ECDH public key. RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the ephemeral-static, ECDH-generated pairwise key-encryption key using the algorithm specified by the KeyWrapAlgorithm.

### 3.1.2. Actions of the sending agent

When using ephemeral-static ECDH with EnvelopedData, the sending agent first obtains the recipient's EC public key and domain parameters (e.g. from the recipient's certificate). The sending agent then determines an integer "keydatalen", which is the KeyWrapAlgorithm symmetric key-size in bits, and also a bit string

"SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 7.2). The sending agent then performs the key deployment and the key agreement operation of the Elliptic Curve Diffie-Hellman Scheme specified in [SP800-56A]. As a result the sending agent obtains:

- an ephemeral public key, which is represented as a value of the type ECPoint (see Section 7.2), encapsulated in a bit string and placed in the KeyAgreeRecipientInfo originator field, and

- a shared secret bit string "K", which is used as the pairwise key-encryption key for that recipient, as specified in [CMS].

### 3.1.3. Actions of the receiving agent

When using ephemeral-static ECDH with EnvelopedData, the receiving agent determines the bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 7.2), and the integer "keydatalen" from the key-size, in bits, of the KeyWrapAlgorithm. The receiving agent retrieves the ephemeral EC public key from the bit string KeyAgreeRecipientInfo originator, with a value of the type ECPoint (see Section 7.2) encapsulated as a bit string, and if present original supplied additional user key material from the ukm field. The receiving agent performs the key agreement operation of the Elliptic Curve Diffie-Hellman Scheme specified in [SP800-56A]. As a result, the receiving agent obtains a shared secret bit string "K", which is used as the pairwise key-encryption key to unwrap the CEK.

## 3.2. EnvelopedData using 1-Pass ECMQV

This section describes how to use the 1-Pass elliptic curve MQV (ECMQV) key agreement algorithm with EnvelopedData, method C(1, 2, ECC MQV) from [SP800-56A]. Like the KEA algorithm [CMS-KEA],





1-Pass ECMQV uses three key pairs: an ephemeral key pair, a static key pair of the sending agent, and a static key pair of the receiving agent. An advantage of using 1-Pass ECMQV is that it can be used with both EnvelopedData and AuthenticatedData.

In an implementation uses 1-Pass ECMQV with CMS EnvelopedData, the following techniques and formats MUST be used.

The fields of EnvelopedData are as in [CMS], as 1-Pass ECMQV is a key agreement algorithm the RecipientInfo kari choice is used. When using 1-Pass ECMQV, the EnvelopedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key. ECC certificates are discussed in Section 5.

### 3.2.1. Fields of KeyAgreeRecipientInfo

When using 1-Pass ECMQV with EnvelopedData, the fields of KeyAgreeRecipientInfo are:

- version MUST be 3.

- originator identifies the static EC public key of the sender. It SHOULD be one of the alternatives, issuerAndSerialNumber or subjectKeyIdentifier, and point to one of the sending agent's certificates.

- ukm MUST be present. The ukm field MUST contain an octet string which is the DER encoding of the type MQVuserKeyingMaterial (see Section 7.2). The MQVuserKeyingMaterial ephemeralPublicKey algorithm field MUST contain the id-ecPublicKey object identifier (see Section 7.1). The parameters associated with id-ecPublicKey MUST be abent or ECPoint. NOTE: The previous version of this document required NULL be present, support is OPTIONAL. The MQVuserKeyingMaterial ephemeralPublicKey publicKey field MUST contain the DER-encoding of the ASN.1 type ECPoint (see Section 7.2) representing the sending agent's ephemeral EC public key. The MQVuserKeyingMaterial addedukm field, if present, SHOULD contain an octet string of additional user keying material of the sending agent.

- keyEncryptionAlgorithm MUST be the key encryption algorithm identifier (see Section 7.1), with the parameters field KeyWrapAlgorithm. The KeyWrapAlgorithm indicates the symmetric encryption algorithm used to encrypt the CEK with the KEK generated using the 1-Pass ECMQV algorithm and any associated parameters. Algorithm requirements are found in Section 8.





- recipientEncryptedKeys contains an identifier and an encrypted key for each recipient. The RecipientEncryptedKey KeyAgreeRecipientIdentifier MUST contain either the

issuerAndSerialNumber identifying the recipient's certificate or the RecipientKeyIdentifier containing the subject key identifier from the recipient's certificate. In both cases, the recipient's certificate contains the recipient's static ECMQV public key. RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the 1-Pass ECMQV-generated pairwise key-encryption key using the algorithm specified by the KeyWrapAlgorithm.

### 3.2.2. Actions of the sending agent

When using 1-Pass ECMQV with EnvelopedData, the sending agent first obtains the recipient's EC public key and domain parameters (e.g. from the recipient's certificate), and checks that the domain parameters are the same, as the sender's domain parameters. The sending agent then determines an integer "keydatalen", which is the KeyWrapAlgorithm symmetric key-size in bits, and also a bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 7.2). The sending agent then performs the key deployment and key agreement operations of the Elliptic Curve MQV Scheme specified in [SP800-56A]. As a result, the sending agent obtains:

- an ephemeral public key, which is represented as a value of type ECPoint (see Section 7.2), encapsulated in a bit string, placed in an MQVuserKeyingMaterial ephemeralPublicKey publicKey field (see Section 7.2), and

- a shared secret bit string "K", which is used as the pairwise key-encryption key for that recipient, as specified in [CMS].

The ephemeral public key can be re-used with an AuthenticatedData for greater efficiency.

### 3.2.3. Actions of the receiving agent

When using 1-Pass ECMQV with EnvelopedData, the receiving agent determines the bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 7.2), and the integer "keydatalen" from the key-size, in bits, of the KeyWrapAlgorithm. The receiving agent then retrieves the static and ephemeral EC public keys of the originator, from the originator and ukm fields as described in Section 3.2.1, and its static EC public key identified in the rid field and checks that the domain parameters are the same. The receiving agent then performs the key agreement operation of the

Elliptic Curve MQV Scheme [SP800-56A]. As a result, the receiving agent obtains a shared secret bit string "K" which is used as the pairwise key-encryption key to unwrap the CEK.

## 4. AuthenticatedData and AuthEnvelopedData using ECC

This section describes how to use ECC algorithms with the CMS AuthenticatedData format. AuthenticatedData lacks non-repudiation, and so in some instances is preferable to SignedData. (For example, the sending agent might not want the message to be authenticated when forwarded.)

This section also describes how to use ECC algorithms with the CMS AuthEnvelopedData format [CMS-AUTHENV]. AuthEnvelopedData supports authentication and encryption, and in some instances is preferable to signing and then encrypting data.

### 4.1. AuthenticatedData using 1-pass ECMQV

This section describes how to use the 1-Pass elliptic curve MQV (ECMQV) key agreement algorithm with AuthenticatedData. ECMQV is method C(1, 2, ECC MQV) from [SP800-56A]. An advantage of using 1-Pass ECMQV is that it can be used with EnvelopedData, AuthenticatedData, and AuthEnvelopedData.

When using ECMQV with AuthenticatedData, the fields of AuthenticatedData are as in [CMS], but with the following restrictions:

- macAlgorithm MUST contain the algorithm identifier of the message authentication code algorithm (see Section 7.1) which MUST be one of the following: id-hmacWithSHA1, id-hmacWITHSHA224, id-hmacWITHSHA256, id-hmacWITHSHA384, and id-hmacWITHSHA512.

- digestAlgorithm MUST contain the algorithm identifier of the hash algorithm (see Section 7.1) which MUST be one of the following: id-sha1, id-sha224, id-sha256, id-sha384, and id-sha512.

The fields of AuthenticatedData are as in [CMS], as 1-Pass ECMQV is a key agreement algorithm the RecipientInfo kari choice is used. When using 1-Pass ECMQV, the AuthenticatedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key. ECC certificates are discussed in Section 5.

### 4.1.1. Fields of the KeyAgreeRecipientInfo

The AuthenticatedData KeyAgreeRecipientInfo fields are used in the same manner as the fields for the corresponding EnvelopedData KeyAgreeRecipientInfo fields of Section 3.2.1 of this document.

### 4.1.2. Actions of the sending agent

The sending agent uses the same actions as for EnvelopedData with 1-Pass ECMQV, as specified in Section 3.2.2 of this document.

The ephemeral public key can be re-used with an EnvelopedData for greater efficiency.

Note: if there are multiple recipients, an attack is possible where one recipient modifies the content without other recipients noticing [BON]. A sending agent who is concerned with such an attack SHOULD use a separate AuthenticatedData for each recipient.

### 4.1.3. Actions of the receiving agent

The receiving agent uses the same actions as for EnvelopedData with 1-Pass ECMQV, as specified in Section 3.2.3 of this document.

Note: see Note in Section 4.1.2.

## 4.2. AuthEnvelopedData using 1-pass ECMQV

This section describes how to use the 1-Pass elliptic curve MQV (ECMQV) key agreement algorithm with AuthEnvelopedData. ECMQV is method C(1, 2, ECC MQV) from [SP800-56A]. An advantage of using 1-Pass ECMQV is that it can be used with EnvelopedData, AuthenticatedData, and AuthEnvelopedData.

The fields of AuthEnvelopedData are as in [CMS], as 1-Pass ECMQV is a key agreement algorithm the RecipientInfo kari choice is used. When using 1-Pass ECMQV, the AuthEnvelopedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key. ECC certificates are discussed in Section 5.

### 4.2.1. Fields of the KeyAgreeRecipientInfo

The AuthEnvelopedData KeyAgreeRecipientInfo fields are used in the same manner as the fields for the corresponding EnvelopedData KeyAgreeRecipientInfo fields of Section 3.2.1 of this document.





### 4.2.2. Actions of the sending agent

The sending agent uses the same actions as for EnvelopedData with 1-Pass ECMQV, as specified in Section 3.2.2 of this document.

The ephemeral public key can be re-used with an EnvelopedData for greater efficiency.

### 4.2.3. Actions of the receiving agent

The receiving agent uses the same actions as for EnvelopedData with 1-Pass ECMQV, as specified in Section 3.2.3 of this document.

## 5. Certificates using ECC

Internet X.509 certificates [PKI] can be used in conjunction with this specification to distribute agents' public keys. The use of ECC algorithms and keys within X.509 certificates is specified in [PKI-ALG].

## 6. SMIMECapabilities Attribute and ECC

A sending agent MAY announce to receiving agents that it supports one or more of the ECC algorithms in this document by using the SMIMECapabilities signed attribute [MSG].

The SMIMECapability value to indicate support for one of the ECDSA signature algorithms is a SEQUENCE with the capabilityID field containing the object identifier ecdsa-with-SHA* object identifiers (where * is 1, 224, 256, 384, or 512) and with NULL parameters. The DER encodings are:

ecdsa-with-SHA1: 30 0b 06 07 2a 86 48 ce 3d 04 01 05 00

ecdsa-with-SHA224: 30 0c 06 08 2a 86 48 ce 3d 04 03 01 05 00

ecdsa-with-SHA256: 30 0c 06 08 2a 86 48 ce 3d 04 03 02 05 00

ecdsa-with-SHA384: 30 0c 06 08 2a 86 48 ce 3d 04 03 03 05 00

ecdsa-with-SHA512: 30 0c 06 08 2a 86 48 ce 3d 04 03 04 05 00





The SMIMECapability value to indicate support for
a) the standard ECDH key agreement algorithm,
b) the cofactor ECDH key agreement algorithm, or
c) the 1-Pass ECMQV key agreement algorithm
is a SEQUENCE with the capabilityID field containing the object identifier
a) dhSinglePass-stdDH-sha*kdf-scheme,
b) dhSinglePass-cofactorDH-sha*kdf-scheme, or
c) mqvSinglePass-sha*kdf-scheme
respectively (where * is 1, 224, 256, 384, or 512) with the parameters present. The parameters indicate the supported key-encryption algorithm with the KeyWrapAlgorithm algorithm identifier.

Example DER encodings that indicate some capabilities are as follows (KA is key agreement, KDF is key derivation function, and Wrap is key wrap algorithm):

KA=ECDH standard KDF=SHA1 Wrap=3DES

```
30 1c
      06 09 2b 81 05 10 86 48 3f 00 02
      30 0f
            06 0b 2a 86 48 86 f7 0d 01 09 10 03 06
            05 00
```

KA=ECDH standard KDF=SHA256 Wrap=AES128

```
30 17
      06 06 2b 81 04 01 0B 01
      30 0d
            06 09 60 86 48 01 65 03 04 01 05
            05 00
```

```
KA=ECDH standard KDF=SHA384 Wrap=AES256

  30 17
        06 06 2b 81 04 01 0B 02
        30 0d
              06 09 60 86 48 01 65 03 04 01 2D
              05 00
```





```
KA=ECDH cofactor KDF=SHA1 Wrap=3DES

  30 1c
        06 09 2b 81 05 10 86 48 3f 00 03
        30 0f
              06 0b 2a 86 48 86 f7 0d 01 09 10 03 06
              05 00

KA=ECDH cofactor KDF=SHA256 Wrap=AES128

  30 17
        06 06 2b 81 04 01 0E 01
        30 0d
              06 09 60 86 48 01 65 03 04 01 05
              05 00

KA=ECDH cofactor KDF=SHA384 Wrap=AES256

  30 17
        06 06 2b 81 04 01 0E 02
        30 0d
              06 09 60 86 48 01 65 03 04 01 2D
              05 00

KA=ECMQV 1-Pass KDF=SHA1 Wrap=3DES

   30 1c
         06 09 2b 81 05 10 86 48 3f 00 10
```

```
            30 0f
                  06 0b 2a 86 48 86 f7 0d 01 09 10 03 06
                  05 00

   KA=ECMQV 1-Pass KDF=SHA256 Wrap=AES128

     30 17
           06 06 2b 81 04 01 0F 01
           30 0d
                 06 09 60 86 48 01 65 03 04 01 05
                 05 00

   KA=ECMQV 1-Pass KDF=SHA384 Wrap=AES256

     30 17
           06 06 2b 81 04 01 0F 02
           30 0d
                 06 09 60 86 48 01 65 03 04 01 2D
                 05 00
```





## 7. ASN.1 Syntax

The ASN.1 syntax used in this document is gathered in this section for reference purposes.

### 7.1. Algorithm Identifiers

This section provides the object identifiers for the algorithms used in this document along with any associated parameters.

#### 7.1.1. Digest Algorithms

Digest algorithm object identifiers are used in the SignedData digestAlgorithms and digestAlgorithm fields, the AuthenticatedData digestAlgorithm field, and the AuthEnvelopedData digestAlgorithm field. The digest algorithms used in this document are: SHA-1, SHA224, SHA-256, SHA-384, and SHA-512. The object identifiers and parameters associated with these algorithms are found in [SMIME-SHA2].

#### 7.1.2. Originator Public Key

The KeyAgreeRecipientInfo originator filed use the following object

identifier to indicate an elliptic curve public key:

```
id-ecPublicKey OBJECT IDENTIFIER ::= {
  ansi-x9-62 keyType(2) 1 }
```

where

```
ansi-x9-62 OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) 10045 }
```

When the object identifier id-ecPublicKey is used here with an algorithm identifier, the associated parameters MUST be either absent or ECPoint. Implementations MUST accept id-ecPublicKey with the parameters field with absent, NULL, and ECPoint parameters. If ECPoint is present its value is ignored. Implementations SHOULD generate absent parameters for the id-ecPublicKey object identifier in the KeyAgreeRecipientInfo originator field.

### 7.1.3. Signature Algorithms

Signature algorithm identifiers are used in the SignedData signatureAlgorithm and signature field. The signature algorithms used in this document are ECDSA with SHA-1, ECDSA with SHA-224, ECDSA with SHA-256, ECDSA with SHA-384, and ECDSA with SHA-512. The object





identifiers and parameters associated with these algorithms are found in [PKI-ALG].

### 7.1.4. Key Agreement Algorithms

Key agreement algorithms are used in EnvelopedData, AuthenticatedData, and AuthEnvelopedData in the KeyAgreeRecipientInfo keyEncryptionAlgorithm field. The following object identifiers indicate the key agreement algorithms used in this document [SP800-56A]:

```
dhSinglePass-stdDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
  x9-63-scheme 2 }

dhSinglePass-stdDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 0 }

dhSinglePass-stdDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
```

```
      secg-scheme 11 1 }

    dhSinglePass-stdDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 11 2 }

    dhSinglePass-stdDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 11 3 }

    dhSinglePass-cofactorDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
      x9-63-scheme 3 }

    dhSinglePass-cofactorDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 14 0 }

    dhSinglePass-cofactorDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 14 1 }

    dhSinglePass-cofactorDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 14 2 }

    dhSinglePass-cofactorDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 14 3 }

    mqvSinglePass-sha1kdf-scheme OBJECT IDENTIFIER ::= {
      x9-63-scheme 16 }

    mqvSinglePass-sha224kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 15 0 }
```





```
    mqvSinglePass-sha256kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 15 1 }

    mqvSinglePass-sha384kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 15 2 }

    mqvSinglePass-sha512kdf-scheme OBJECT IDENTIFIER ::= {
      secg-scheme 15 3 }

  where

    x9-63-scheme OBJECT IDENTIFIER ::= {
      iso(1) identified-organization(3) tc68(133) country(16)
```

```
      x9(840) x9-63(63) schemes(0) }
```

and

```
   secg-scheme OBJECT IDENTIFIER ::= {
     iso(1) identified-organization(3) certicom(132) schemes(1) }
```

When the object identifiers are used here within an algorithm identifier, the associated parameters field contains KeyWrapAlgorithm to indicate the key wrap algorithm and any associated parameters.

### 7.1.5. Key Wrap Algorithms

Key wrap algorithms are used as part of the parameters in the key agreement algorithm. The key wrap algorithms used in this document are Triple-DES, AES-128, AES-192, AES-256. The object identifier and parameters for these algorithms are found in [CMS-ALG] and [CMS-AES].

### 7.1.6. Content Encryption Algorithms

Content encryption algorithms are used in EnvelopedData and AuthEnvelopedData in the EncryptedContentInfo contentEncryptionAlgorithm field. The content encryption algorithms used with EnvelopedData in this document are AES-128 in CBC mode, AES-192 in CBC mode, and AES-256 in CBC mode. The object identifiers and parameters associated with these algorithms are found in [CMS-AES]. The content encryption algorithms used with AuthEnvelopedData in this document are AES-128 in CCM mode, AES-192 in CCM mode, AES-256 in CCM mode, AES-128 in GCM mode, AES-192 in GCM mode, and AES-256 in GCM mode. The object identifiers and parameters associated with these algorithms are found in [CMS-AESCG].





### 7.1.7. Message Authentication Code Algorithms

Message authentication code algorithms are used in AuthenticatedData and AuthEnvelopedData in the macAlgorithm field. The message authentication code algorithms used in this document are HMAC with SHA-1, HMAC with SHA-224, HMAC with SHA-1, HMAC with SHA-1, and HMAC with SHA-1. The object identifiers and parameters associated with these algorithms are found in [HMAC-SHA1] and [HMAC-SHA2].

## 7.2. Other Syntax

The following additional syntax is used here.

When using ECDSA with SignedData, ECDSA signatures are encoded using the type:

```
  ECDSA-Sig-Value ::= SEQUENCE {
    r INTEGER,
    s INTEGER }
```

ECDSA-Sig-Value is specified in [PKI-ALG]. Within CMS, ECDSA-Sig-Value is DER-encoded and placed within a signature field of SignedData.

When using ECDH and ECMQV with EnvelopedData, AuthenticatedData, and AuthEnvelopedData, ephemeral and static public keys are encoded using the type ECPoint. Implementations MUST support uncompressed keys and MAY support compressed keys.

```
  ECPoint ::= OCTET STRING
```

When using ECMQV with EnvelopedData, AuthenticatedData, and AuthEnvelopedData, the sending agent's ephemeral public key and additional keying material are encoded using the type:

```
  MQVuserKeyingMaterial ::= SEQUENCE {
    ephemeralPublicKey      OriginatorPublicKey,
    addedukm            [0] EXPLICIT UserKeyingMaterial OPTIONAL  }
```

The ECPoint syntax in used to represent the ephemeral public key and placed in the ephemeralPublicKey field. The additional user keying material is placed in the addedukm field. Then the MQVuserKeyingMaterial value is DER-encoded and placed within a ukm field of EnvelopedData, AuthenticatedData, or AuthEnvelopedData.





When using ECDH or ECMQV with EnvelopedData, AuthenticatedData, or AuthEnvelopedData, the key-encryption keys are derived by using the type:

```
ECC-CMS-SharedInfo ::= SEQUENCE {
  keyInfo         AlgorithmIdentifier,
  entityUInfo [0] EXPLICIT OCTET STRING OPTIONAL,
  suppPubInfo [2] EXPLICIT OCTET STRING  }
```

The fields of ECC-CMS-SharedInfo are as follows:

keyInfo contains the object identifier of the key-encryption algorithm (used to wrap the CEK) and associated parameters. In this specification, 3DES wrap has NULL parameters while the AES wraps have absent parameters.

entityUInfo optionally contains additional keying material supplied by the sending agent. When used with ECDH and CMS, the entityUInfo field contains the octet string ukm. When used with ECMQV and CMS, the entityUInfo contains the octet string addedukm (encoded in MQVuserKeyingMaterial).

suppPubInfo contains the length of the generated KEK, in bits, represented as a 32 bit number, as in [CMS-DH] and [CMS-AES]. (E.g. for AES-256 it would be 00 00 01 00.)

Within CMS, ECC-CMS-SharedInfo is DER-encoded and used as input to the key derivation function, as specified in [SP800-56A].

Note that ECC-CMS-SharedInfo differs from the OtherInfo specified in [CMS-DH]. Here, a counter value is not included in the keyInfo field because the key derivation function specified in [SP800-56A] ensures that sufficient keying data is provided.

## 8. Recommended Algorithms and Elliptic Curves

It is RECOMMEND that implementations of this specification support SignedData. Support for EnvelopedData and AuthenticatedData is OPTIONAL.

In order to encourage interoperability, implementations SHOULD use the elliptic curve domain parameters specified by [PKI-ALG].

Implementations that support SignedData with ECDSA:

- MUST support ECDSA with SHA-256.

- MAY support ECDSA with SHA-1, ECDSA with SHA-224, ECDSA with SHA-384, and ECDSA with SHA-512.

When using ECDSA, it is RECOMMENDED that the P-224 curve be used with SHA-224, the P-256 curve be used with SHA-256, the P-384 curve be used with SHA-384, and the P-521 curve be used with SHA-512.

If EnvelopedData is supported, then ephemeral-static ECDH standard primitive MUST be supported.

Implementations that support EnvelopedData with the ephemeral-static ECDH standard primitive:

- MUST support the dhSinglePass-stdDH-sha256kdf-scheme key agreement algorithm, the id-aes128-wrap key wrap algorithm, and the id-aes128-cbc content encryption algorithm

- MAY support the dhSinglePass-stdDH-sha1kdf-scheme, dhSinglePass-stdDH-sha224kdf-scheme, dhSinglePass-stdDH-sha384kdf-scheme and dhSinglePass-stdDH-sha512kdf-scheme key agreement algorithms, the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms and the id-aes192-cbc and id-aes256-cbc content encryption algorithms.

Implementations that support EnvelopedData with the ephemeral-static ECDH cofactor primitive:

- MUST support the dhSinglePass-cofactorDH-sha256kdf-scheme key agreement algorithm, the id-aes128-wrap key wrap algorithm, and the id-aes128-cbc content encryption algorithm.

- MAY support the dhSinglePass-cofactorDH-sha1kdf-scheme, dhSinglePass-cofactorDH-sha224kdf-scheme, dhSinglePass-cofactorDH-sha384kdf-scheme, and dhSinglePass-cofactorDH-sha512kdf-scheme key agreement, the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms and the id-aes192-cbc and id-aes256-cbc content encryption algorithms.

Implementations that support EnvelopedData with 1-Pass ECMQV:

- MUST support the mqvSinglePass-sha256kdf-scheme key agreement algorithm, the id-aes128-wrap key wrap algorithm, and the id-aes128-cbc content encryption algorithm.

- MAY support mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, and mqvSinglePass-sha512kdf-scheme key agreement algorithms, the id-

alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms and the id-aes192-cbc and id-aes256-cbc content encryption algorithms.

Implementations that support AuthenticatedData with 1-Pass ECMQV:

- MUST support the mqvSinglePass-sha256kdf-scheme key agreement, the id-aes128-wrap key wrap, and the id-aes128-cbc content encryption, the id-sha256 message digest, and id-hmacWithSHA256 message authentication code algorithms.

- MAY support the mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, mqvSinglePass-sha512kdf-scheme key agreement algorithms, the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms, the id-aes192-cbc and id-aes256-cbc content encryption algorithms, the id-sha1, id-sha224, id-sha384, and id-sha512, message digest algorithms, and the id-hmacWithSHA1, id-hmacWithSHA224, id-hmacWithSHA384, id-hmacWithSHA512 message authentication code algorithms.

Implementations that support AuthEnvelopedData with 1-Pass ECMQV:

- MUST support the mqvSinglePass-sha256kdf-scheme key agreement, the id-aes128-wrap key wrap, the id-aes128-ccm authenticated-content encryption, the id-sha256 message digest, and the id-hmacWithSHA256 message authentication cod algorithms.

- MAY support the mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, and mqvSinglePass-sha512kdf-scheme key agreement algorithms, the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms, the id-aes192-ccm and id-aes256-ccm authenticated-content encryption algorithms, the id-sha1, id-sha224, id-sha384, and id-sha512, message digest algorithms, and id-hmacWithSHA1, id-hmacWithSHA224, id-hmacWithSHA384, id-hmacWithSHA512 message authentication code algorithms.

## 9. Security Considerations

Cryptographic algorithms will be broken or weakened over time. Implementers and users need to check that the cryptographic algorithms listed in this document continue to provide the expected

level of security. The IETF from time to time may issue documents dealing with the current state of the art.





Cryptographic algorithms rely on random number. See [RANDOM] for guidance on generation of random numbers.

Receiving agents that validate signatures and sending agents that encrypt messages, need to be cautious of cryptographic processing usage when validating signatures and encrypting messages using keys larger than those mandated in this specification. An attacker could send certificates with keys which would result in excessive cryptographic processing, for example keys larger than those mandated in this specification, which could swamp the processing element. Agents which use such keys without first validating the certificate to a trust anchor are advised to have some sort of cryptographic resource management system to prevent such attacks.

Using secret keys of an appropriate size is crucial to the security of a Diffie-Hellman exchange. For elliptic curve groups, the size of the secret key must be equal to the size of n (the order of the group generated by the point g). Using larger secret keys provides absolutely no additional security, and using smaller secret keys is likely to result in dramatically less security. (See [SP800-56A] for more information on selecting secret keys.)

This specification is based on [CMS], [CMS-AUTHENV], [CMS-ALG], [CMS-AESCG], [X9.62], [SEC1], and [SEC2] and the appropriate security considerations of those documents apply.

In addition, implementors of AuthenticatedData should be aware of the concerns expressed in [BON] when using AuthenticatedData to send messages to more than one recipient. Also, users of MQV should be aware of the vulnerability in [K].

When implementing EnvelopedData, AuthenticatedData, and AuthEnvelopedData, there are five algorithm related choices that need to be made:

1) What is the public key size?
2) What is the KDF?
3) What is the key wrap algorithm?

4) What is the content encryption algorithm?
5) What is the curve?

Consideration must be given to strength of the security provided by each of these choices. Security is measured in bits, where a strong symmetric cipher with a key of X bits is said to provide X bits of security. It is recommended that the bits of security provided by each are roughly equivalent. The following table provides comparable minimum bits of security [SP800-57] for the ECDH/ECMQV key sizes,





KDFs, key wrapping algorithms, and content encryption algorithms. It also lists curves [PKI-ALG] for the key sizes.

| Minimum Bits of Security | ECDH or ECQMV Key Size | Key Derivation Function | Key Wrap Alg. | Content Encryption Alg. | Curves |
|---|---|---|---|---|---|
| 80 | 160-223 | SHA1 | 3DES | 3DES CBC | sect163k1 |
| | | SHA224 | AES-128 | AES-128 CBC | secp163r2 |
| | | SHA256 | AES-192 | AES-192 CBC | secp192r1 |
| | | SHA384 | AES-256 | AES-256 CBC | |
| | | SHA512 | | | |
| 112 | 224-255 | SHA1 | 3DES | 3DES CBC | secp224r1 |
| | | SHA224 | AES-128 | AES-128 CBC | sect233k1 |
| | | SHA256 | AES-192 | AES-192 CBC | sect233r1 |
| | | SHA384 | AES-256 | AES-256 CBC | |
| | | SHA512 | | | |
| 128 | 256-383 | SHA1 | AES-128 | AES-128 CBC | secp256r1 |
| | | SHA224 | AES-192 | AES-192 CBC | sect283k1 |
| | | SHA256 | AES-256 | AES-256 CBC | sect283r1 |
| | | SHA384 | | | |
| | | SHA512 | | | |
| 192 | 384-511 | SHA224 | AES-192 | AES-192 CBC | secp384r1 |
| | | SHA256 | AES-256 | AES-256 CBC | sect409k1 |
| | | SHA384 | | | sect409r1 |
| | | SHA512 | | | |
| 256 | 512+ | SHA256 | AES-256 | AES-256 CBC | secp521r1 |
| | | SHA384 | | | sect571k1 |
| | | SHA512 | | | sect571r1 |

To promote interoperability, the following choices are RECOMMENDED:

| Minimum Bits of Security | ECDH or ECQMV Key Size | Key Derivation Function | Key Wrap Alg. | Content Encryption Alg. | Curve |
|---|---|---|---|---|---|
| 80 | 192 | SHA256 | 3DES | 3DES CBC | secp192r1 |
| 112 | 224 | SHA256 | 3DES | 3DES CBC | secp224r1 |
| 128 | 256 | SHA256 | AES-128 | AES-128 CBC | secp256r1 |
| 192 | 384 | SHA384 | AES-256 | AES-256 CBC | secp384r1 |
| 256 | 512 | SHA512 | AES-256 | AES-256 CBC | secp521r1 |

When implementing SignedData, there are three algorithm related choices that need to be made:

1) What is the public key size?
2) What is the hash algorithm?
3) What is the curve?

Consideration must be given to the bits of security provided by each of these choices. Security is measured in bits, where a strong symmetric cipher with a key of X bits is said to provide X bits of security. It is recommended that the bits of security provided by

each choice are roughly equivalent. The following table provides comparable minimum bits of security [SP800-57] for the ECDSA key sizes and message digest algorithms. It also lists curves [PKI-ALG] for the key sizes.





| Minimum Bits of Security | ECDSA Key Size | Message Digest Algorithm | Curve |
|---|---|---|---|
| 80 | 160-223 | SHA1 | sect163k1 |
| | | SHA224 | secp163r2 |
| | | SHA256 | secp192r1 |
| | | SHA384 | |
| | | SHA512 | |
| 112 | 224-255 | SHA224 | secp224r1 |
| | | SHA256 | sect233k1 |
| | | SHA384 | sect233r1 |
| | | SHA512 | |
| 128 | 256-383 | SHA256 | secp256r1 |
| | | SHA384 | sect283k1 |
| | | SHA512 | sect283r1 |
| 192 | 384-511 | SHA384 | secp384r1 |
| | | SHA512 | sect409k1 |
| | | | sect409r1 |

| | | | |
|---|---|---|---|
| 256 | 512+ | SHA512 | secp521r1<br>sect571k1<br>sect571r1 |

To promote interoperability, the following choices are RECOMMENDED:

| Minimum Bits of Security | ECDSA Key Size | Message Digest Algorithm | Curve |
|---|---|---|---|
| 80 | 192 | SHA256 | sect192r1 |
| 112 | 224 | SHA256 | secp224r1 |
| 128 | 256 | SHA256 | secp256r1 |
| 192 | 384 | SHA384 | secp384r1 |
| 256 | 512+ | SHA512 | secp521r1 |





## 10. IANA Considerations

None.

## 11. References

### 11.1. Normative

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 3852, July 2004.

[CMS-AES] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.

[CMS-AESCG] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, November 2007.

[CMS-ALG] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[CMS-ASN] Hoffman, P., and J. Schaad, "New ASN.1 Modules for CMS", draft-ietf-smime-new-asn1, work-in-progress.

[CMS-AUTHENV] Housley, R. "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.

[CMS-DH] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[FIPS180-3] National Institute of Standards and Technology (NIST), FIPS Publication 180-3: Secure Hash Standard, June 2003.

[FIPS186-3] National Institute of Standards and Technology (NIST), FIPS Publication 186-3: Digital Signature Standard, March 2006.

[HMAC-SHA1] Krawczyk, M., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[HMAC-SHA2] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", RFC 4231, December 2005.





[MUST] Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[MSG] Ramsdell, B., and S. Turner, "S/MIME Version 3.2 Message Specification", draft-ietf-smime-3851bis, work-in-progress.

[PKI] Cooper, D., Santesson, S., Farrell, S., Boeyen, S. Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[PKI-ALG] Turner, S., Brown, D., Yiu, K., Housley, R., and W.

Polk, "Elliptic Curve Cryptography Subject Public Key Information", draft-ietf-pkix-ecc-subpubkeyinfo, work-in-progress.

[PKI-ASN] Hoffman, P., and J. Schaad, "New ASN.1 Modules for PKIX", draft-ietf-pkix-new-asn1, work-in-progress.

[RANDOM] Eastlake 3rd, D., Crocker, S., and J. Schiller, "Randomness Recommendations for Security", RFC 4086, June 2005.

[RSAOAEP] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[SEC1] SECG, "Elliptic Curve Cryptography", Standards for Efficient Cryptography Group, 2000. Available from www.secg.org/collateral/sec1.pdf.

[SEC2] SECG, "Recommended Elliptic Curve Domain Parameters", Standards for Efficient Cryptography Group, 2000. Available from www.secg.org/collateral/sec2.pdf.

[SMIME-SHA2] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", work-in-progress.

[SP800-56A] National Institute of Standards and Technology (NIST), Special Publication 800-56A: Recommendation Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised), March 2007.





[X9.62] American National Standards Institute (ANSI), ANS X9.62-2005: The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.

[X.208] ITU-T Recommendation X.208 (1988) | ISO/IEC 8824-1:1988. Specification of Abstract Syntax Notation One (ASN.1).

[X.680] ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1 :2002. Information Technology - Abstract Syntax Notation One.

[X.681] ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-2 :2002. Information Technology - Abstract Syntax Notation One: Information Object Specification.

[X.682] ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3 :2002. Information Technology - Abstract Syntax Notation One: Constraint Specification.

[X.683] ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002. Information Technology - Abstract Syntax Notation One: Parameterization of ASN.1 Specifications, 2002.

## 11.2. Informative

[BON] D. Boneh, "The Security of Multicast MAC", Presentation at Selected Areas of Cryptography 2000, Center for Applied Cryptographic Research, University of Waterloo, 2000. Paper version available from http://crypto.stanford.edu/~dabo/papers/mmac.ps

[CMS-KEA] Pawling, J., "CMS KEA and SKIPJACK Conventions", RFC 2876, July 2000.

[K] B. Kaliski, "MQV Vulnerability", Posting to ANSI X9F1 and IEEE P1363 newsgroups, 1998.

[SP800-57] National Institute of Standards and Technology (NIST), Special Publication 800-57: Recommendation for Key Management, August 2005.





# Appendix A ASN.1 Modules

Appendix A.1 provides the normative ASN.1 definitions for the

structures described in this specification using ASN.1 as defined in [X.208].

Appendix A.2 provides an informative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.680], [X.681], [X.682], [X.683]. This appendix contains the same information as Appendix A.1 in a more recent (and precise) ASN.1 notation, however Appendix A.1 takes precedence in case of conflict.

## Appendix A.1 1988 ASN.1 Module

```
SMIMEECCAlgs-1988
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    smime(16) modules(0) TBD }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS

-- From [PKI]

AlgorithmIdentifier
  FROM PKIX1Explicit88
    { iso(1) identified-organization(3) dod(6)
      internet(1) security(5) mechanisms(5) pkix(7) mod(0)
      pkix1-explicit(18) }

-- From [RSAOAEP]

id-sha224, id-sha256, id-sha384, id-sha512
  FROM PKIX1-PSS-OAEP-Algorithms
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-rsa-pkalgs(33) }
```

```
-- From [PKI-ALG]

id-sha1, ecdsa-with-SHA1, ecdsa-with-SHA224,
ecdsa-with-SHA256, ecdsa-with-SHA384, ecdsa-with-SHA512,
id-ecPublicKey, ECDSA-Sig-Value, ECPoint
  FROM PKIXAlgs-1988
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0) TBD }

-- From [CMS]

OriginatorPublicKey, UserKeyingMaterial
  FROM CryptographicMessageSyntax2004
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) cms-2004(24) }

-- From [CMS-ALG]

hMAC-SHA1, des-ede3-cbc, id-alg-CMS3DESwrap, CBCParameter
  FROM CryptographicMessageSyntaxAlgorithms
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) cmsalg-2001(16) }

-- From [CMS-AES]

id-aes128-CBC, id-aes192-CBC, id-aes256-CBC, AES-IV,
id-aes128-wrap, id-aes192-wrap, id-aes256-wrap
  FROM CMSAesRsaesOaep
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cms-aes(19) }

-- From [CMS-AESCG]

id-aes128-CCM, id-aes192-CCM, id-aes256-CCM, CCMParameters
id-aes128-GCM, id-aes192-GCM, id-aes256-GCM, GCMParameters
  FROM CMS-AES-CCM-and-AES-GCM
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cms-aes(32) }

;

--
-- ECDSA with SHA-2 Algorithms
--

-- ecdsa-with-SHA1 Parameters are NULL
```

```
   -- ecdsa-with-SHA224 Parameters are ABSENT

   -- ecdsa-with-SHA256 Parameters are ABSENT

   -- ecdsa-with-SHA384 Parameters are ABSENT

   -- ecdsa-with-SHA512 Parameters are absent

   -- ECDSA Signature Value
   -- Contents of SignatureValue OCTET STRING

   -- ECDSA-Sig-Value ::= SEQUENCE {
   --   r  INTEGER,
   --   s  INTEGER
   -- }

   --
   -- Key Agreement Algorithms
   --

   x9-63-scheme OBJECT IDENTIFIER ::= {
     iso(1) identified-organization(3) tc68(133) country(16) x9(840)
     x9-63(63) schemes(0) }

   secg-scheme OBJECT IDENTIFIER ::= {
     iso(1) identified-organization(3) certicom(132) schemes(1) }

   --
   -- Diffie-Hellman Single Pass, Standard, with KDFs
   --

   -- Parameters are always present and indicate the key wrap algorithm
   -- with KeyWrapAlgorithm

   dhSinglePass-stdDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 2 }

   dhSinglePass-stdDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 11 0 }

   dhSinglePass-stdDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 11 1 }
```

```
   dhSinglePass-stdDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 11 2 }
```





```
   dhSinglePass-stdDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 11 3 }

   --
   -- Diffie-Hellman Single Pass, Cofactor, with KDFs
   --

   dhSinglePass-cofactorDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 3 }

   dhSinglePass-cofactorDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 0 }

   dhSinglePass-cofactorDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 1 }

   dhSinglePass-cofactorDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 2 }

   dhSinglePass-cofactorDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 3 }

   --
   -- MQV Single Pass, Cofactor, with KDFs
   --

   mqvSinglePass-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 16 }

   mqvSinglePass-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 0 }

   mqvSinglePass-sha256kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 1 }

   mqvSinglePass-sha384kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 2 }
```

```
   mqvSinglePass-sha512kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 3 }
```





```
   --
   -- Key Wrap Algorithms
   --

   KeyWrapAlgorithm ::= AlgorithmIdentifier

   -- id-alg-CMS3DESwrap Parameters are NULL
   -- id-aes128-wrap Parameters are ABSENT
   -- id-aes192-wrap Parameters are ABSENT
   -- id-aes256-wrap Parameters are ABSENT

   --
   -- Content Encryption Algorithms
   --

   -- des-ede3-cbc Parameters are CBCParameter
   -- id-aes128-CBC Parameters are AES-IV
   -- id-aes192-CBC Parameters are AES-IV
   -- id-aes256-CBC Parameters are AES-IV
   -- id-aes128-CCM Parameters are CCMParameters
   -- id-aes192-CCM Parameters are CCMParameters
   -- id-aes256-CCM Parameters are CCMParameters
   -- id-aes128-GCM Parameters are GCMParameters
   -- id-aes192-GCM Parameters are GCMParameters
   -- id-aes256-GCM Parameters are GCMParameters

   --
   -- Message Digest Algorithms
   --

   -- HMAC with SHA-224, HMAC with SHA-256, HMAC with SHA-384,
   -- HMAC with SHA-512 are specified in [HMAC-SHA2]
   -- Parameters are ABSENT
```

```
   -- hMACWithSHA1

   id-hmacWithSHA224 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 8 }

   id-hmacWithSHA256 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 9 }

   id-hmacWithSHA384 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 10 }
```





```
   id-hmacWithSHA512 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 11 }

   --
   -- Originator Public Key Algorithms
   --

   -- id-ecPublicKey Parameters are NULL

   -- Format for both ephemeral and static public keys

   -- ECPoint ::= OCTET STRING

   -- Format of KeyAgreeRecipientInfo ukm field when used with
   -- ECMQV

   MQVuserKeyingMaterial ::= SEQUENCE {
     ephemeralPublicKey       OriginatorPublicKey,
     addedukm             [0] EXPLICIT UserKeyingMaterial OPTIONAL
   }

   -- 'SharedInfo' for input to KDF when using ECDH and ECMQV with
   -- EnvelopedData, AuthenticatedData, or AuthEnvelopedData

   ECC-CMS-SharedInfo ::= SEQUENCE {
     keyInfo         AlgorithmIdentifier,
     entityUInfo [0] EXPLICIT OCTET STRING OPTIONAL,
```

```
  suppPubInfo [2] EXPLICIT OCTET STRING
}

--
-- S/MIME Capabilities
--

--
-- S/MIME Capabilities: ECDSA with SHA1 and SHA2 Algorithms
--

-- ecdsa-with-SHA1 Type NULL
-- ecdsa-with-SHA224 Type NULL
-- ecdsa-with-SHA256 Type NULL
-- ecdsa-with-SHA384 Type NULL
-- ecdsa-with-SHA512 Type NULL
```





```
--
-- S/MIME Capabilities: ECDH, Single Pass, Standard
--

-- dhSinglePass-stdDH-sha1kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-stdDH-sha224kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-stdDH-sha256kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-stdDH-sha384kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-stdDH-sha512kdf Type is the KeyWrapAlgorithm


--
-- S/MIME Capabilities: ECDH, Single Pass, Cofactor
--

-- dhSinglePass-cofactorDH-sha1kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-cofactorDH-sha224kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-cofactorDH-sha256kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-cofactorDH-sha384kdf Type is the KeyWrapAlgorithm
-- dhSinglePass-cofactorDH-sha512kdf Type is the KeyWrapAlgorithm

--
-- S/MIME Capabilities: ECMQV, Single Pass, Standard
```

```
--

-- mqvSinglePass-sha1kdf Type is the KeyWrapAlgorithm
-- mqvSinglePass-sha224kdf Type is the KeyWrapAlgorithm
-- mqvSinglePass-sha256kdf Type is the KeyWrapAlgorithm
-- mqvSinglePass-sha384kdf Type is the KeyWrapAlgorithm
-- mqvSinglePass-sha512kdf Type is the KeyWrapAlgorithm

END
```





Appendix A.2 2004 ASN.1 Module

```
SMIMEECCAlgs-2008
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    smime(16) modules(0) TBD }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS

-- FROM [PKI-ASN]

KEY-WRAP, SIGNATURE-ALGORITHM, DIGEST-ALGORITHM, ALGORITHM,
PUBLIC-KEY, MAC-ALGORITHM, CONTENT-ENCRYPTION, KEY-AGREE
```

```
  FROM AlgorithmInformation
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-algorithInformation(TBD)}

-- From [PKI-ASN]

mda-sha1, sa-ecdsaWithSHA1, sa-ecdsaWithSHA224, sa-ecdsaWithSHA256,
sa-ecdsaWithSHA384, sa-ecdsaWithSHA512, id-ecPublicKey,
ECDSA-Sig-Value, ECPoint
  FROM PKIXAlgs-2008
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0) TBD }

-- From [PKI-ASN]

mda-sha224, mda-sha256, mda-sha384, mda-sha512
  FROM PKIX1-PSS-OAEP-Algorithms
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0) TBD }

-- From [CMS]

OriginatorPublicKey, UserKeyingMaterial
  FROM CryptographicMessageSyntax2004
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) cms-2004(24) }
```





```
-- From [CMS-ASN]

maca-hMAC-SHA1, cea-des-ede3-cbc, kwa-3DESWrap, CBCParameter
  FROM CryptographicMessageSyntaxAlgorithms
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) cmsalg-2001(16) }

-- From [CMS-ASN]

cea-aes128-CBC, cea-aes192-CBC, cea-aes256-CBC, kwa-aes128-wrap,
kwa-aes192-wrap, kwa-aes256-wrap
  FROM CMSAesRsaesOaep
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
```

```
      smime(16) modules(0) id-mod-cms-aes(19) }

-- From [CMS-ASN]

cea-aes128-ccm, cea-aes192-ccm, cea-aes256-ccm, cea-aes128-gcm,
cea-aes192-gcm, cea-aes256-gcm
  FROM CMS-AES-CCM-and-AES-GCM
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) cms-aes-ccm-and-gcm(32) }

;

-- Constrains the SignedData digestAlgorithms field
-- Constrains the SignedData SignerInfo digestAlgorithm field
-- Constrains the AuthenticatedData digestAlgorithm field

-- MessageDigestAlgorithms DIGEST-ALGORITHM ::= {
--  mda-sha1   |
--  mda-sha224 |
--  mda-sha256 |
--  mda-sha384 |
--  mda-sha512,
--  ... -- Extensible
-- }
```





```
-- Constrains the SignedData SignerInfo signatureAlgorithm field

-- SignatureAlgorithms SIGNATURE-ALGORITHM ::= {
--  sa-ecdsaWithSHA1   |
--  sa-ecdsaWithSHA224 |
--  sa-ecdsaWithSHA256 |
--  sa-ecdsaWithSHA384 |
--  sa-ecdsaWithSHA512 ,
```

```
--  ... -- Extensible
-- }

-- ECDSA Signature Value
-- Contents of SignatureValue OCTET STRING

ECDSA-Sig-Value ::= SEQUENCE {
  r  INTEGER,
  s  INTEGER
}

--
-- Key Agreement Algorithms
--

-- Constrains the EnvelopedData RecipientInfo KeyAgreeRecipientInfo
-- keyEncryption Algorithm field
-- Constrains the AuthenticatedData RecipientInfo
-- KeyAgreeRecipientInfo keyEncryption Algorithm field
-- Constrains the AuthEnvelopedData RecipientInfo
-- KeyAgreeRecipientInfo keyEncryption Algorithm field

-- DH variants are not used with AuthenticatedData or
-- AuthEnvelopedData
```





```
KeyAgreementAlgorithms KEY-AGREE ::= {
  kaa-dhSinglePass-stdDH-sha1kdf        |
  kaa-dhSinglePass-stdDH-sha224kdf      |
```

```
    kaa-dhSinglePass-stdDH-sha256kdf      |
    kaa-dhSinglePass-stdDH-sha384kdf      |
    kaa-dhSinglePass-stdDH-sha512kdf      |
    kaa-dhSinglePass-cofactorDH-sha1kdf   |
    kaa-dhSinglePass-cofactorDH-sha224kdf |
    kaa-dhSinglePass-cofactorDH-sha256kdf |
    kaa-dhSinglePass-cofactorDH-sha384kdf |
    kaa-dhSinglePass-cofactorDH-sha512kdf |
    kaa-mqvSinglePass-sha1kdf             |
    kaa-mqvSinglePass-sha224kdf           |
    kaa-mqvSinglePass-sha256kdf           |
    kaa-mqvSinglePass-sha384kdf           |
    kaa-mqvSinglePass-sha512kdf,
    ... -- Extensible
  }

  x9-63-scheme OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) tc68(133) country(16) x9(840)
    x9-63(63) schemes(0) }

  secg-scheme OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) certicom(132) schemes(1) }

  --
  -- Diffie-Hellman Single Pass, Standard, with KDFs
  --

  -- Parameters are always present and indicate the Key Wrap Algorithm

  kaa-dhSinglePass-stdDH-sha1kdf KEY-AGREE ::= {
    IDENTIFIER dhSinglePass-stdDH-sha1kdf-scheme
    PARAMS TYPE KeyWrapAlgorithm ARE required
    UKM IS preferredPresent
  }

  dhSinglePass-stdDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
    x9-63-scheme 2 }

  kaa-dhSinglePass-stdDH-sha224kdf KEY-AGREE ::= {
    IDENTIFIER dhSinglePass-stdDH-sha224kdf-scheme
    PARAMS TYPE KeyWrapAlgorithm ARE required
    UKM IS preferredPresent
  }
```

```
dhSinglePass-stdDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 0 }

kaa-dhSinglePass-stdDH-sha256kdf KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha256kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM IS preferredPresent
}

dhSinglePass-stdDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 1 }

kaa-dhSinglePass-stdDH-sha384kdf KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha384kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM IS preferredPresent
}

dhSinglePass-stdDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 2 }

kaa-dhSinglePass-stdDH-sha512kdf KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha512kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM IS preferredPresent
}

dhSinglePass-stdDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 3 }

--
-- Diffie-Hellman Single Pass, Cofactor, with KDFs
--

kaa-dhSinglePass-cofactorDH-sha1kdf KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-cofactorDH-sha1kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM IS preferredPresent
}

dhSinglePass-cofactorDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
  x9-63-scheme 3 }
```

```
   kaa-dhSinglePass-cofactorDH-sha224kdf KEY-AGREE ::= {
     IDENTIFIER dhSinglePass-cofactorDH-sha224kdf-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
     UKM IS preferredPresent
   }

   dhSinglePass-cofactorDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 0 }

   kaa-dhSinglePass-cofactorDH-sha256kdf KEY-AGREE ::= {
     IDENTIFIER dhSinglePass-cofactorDH-sha256kdf-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
     UKM IS preferredPresent
   }

   dhSinglePass-cofactorDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 1 }

   kaa-dhSinglePass-cofactorDH-sha384kdf KEY-AGREE ::= {
     IDENTIFIER dhSinglePass-cofactorDH-sha384kdf-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
     UKM IS preferredPresent
   }

   dhSinglePass-cofactorDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 2 }

   kaa-dhSinglePass-cofactorDH-sha512kdf KEY-AGREE ::= {
     IDENTIFIER dhSinglePass-cofactorDH-sha512kdf-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
     UKM IS preferredPresent
   }

   dhSinglePass-cofactorDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 3 }

   --
   -- MQV Single Pass, Cofactor, with KDFs
   --

   kaa-mqvSinglePass-sha1kdf KEY-AGREE ::= {
     IDENTIFIER mqvSinglePass-sha1kdf-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
```

```
    UKM IS preferredPresent
  }
```





```
  mqvSinglePass-sha1kdf-scheme OBJECT IDENTIFIER ::= {
    x9-63-scheme 16 }

  kaa-mqvSinglePass-sha224kdf KEY-AGREE ::= {
    IDENTIFIER mqvSinglePass-sha224kdf-scheme
    PARAMS TYPE KeyWrapAlgorithm ARE required
    UKM IS preferredPresent
  }

  mqvSinglePass-sha224kdf-scheme OBJECT IDENTIFIER ::= {
    secg-scheme 15 0 }

  kaa-mqvSinglePass-sha256kdf KEY-AGREE ::= {
    IDENTIFIER mqvSinglePass-sha256kdf-scheme
    PARAMS TYPE KeyWrapAlgorithm ARE required
    UKM IS preferredPresent
  }

  mqvSinglePass-sha256kdf-scheme OBJECT IDENTIFIER ::= {
    secg-scheme 15 1 }

  kaa-mqvSinglePass-sha384kdf KEY-AGREE ::= {
    IDENTIFIER mqvSinglePass-sha384kdf-scheme
    PARAMS TYPE KeyWrapAlgorithm ARE required
    UKM IS preferredPresent
  }

  mqvSinglePass-sha384kdf-scheme OBJECT IDENTIFIER ::= {
    secg-scheme 15 2 }

  kaa-mqvSinglePass-sha512kdf KEY-AGREE ::= {
    IDENTIFIER mqvSinglePass-sha512kdf-scheme
    PARAMS TYPE KeyWrapAlgorithm ARE required
    UKM IS preferredPresent
  }

  mqvSinglePass-sha512kdf-scheme OBJECT IDENTIFIER ::= {
    secg-scheme 15 3 }
```

```
   --
   -- Key Wrap Algorithms
   --

   KeyWrapAlgorithm KEY-WRAP ::= {
     kwa-3des   |
     kwa-aes128 |
     kwa-aes192 |
     kwa-aes256,
     ... -- Extensible
   }

   --
   -- Content Encryption Algorithms
   --

   -- Constrains the EnvelopedData EncryptedContentInfo encryptedContent
   -- field and the AuthEnvelopedData EncryptedContentInfo
   -- contentEncryptionAlgorithm field

   -- ContentEncryptionAlgorithms CONTENT-ENCRYPTION ::= {
   --   cea-des-ede3-cbc |
   --   cea-aes128-cbc   |
   --   cea-aes192-cbc   |
   --   cea-aes256-cbc   |
   --   cea-aes128-ccm   |
   --   cea-aes192-ccm   |
   --   cea-aes256-ccm   |
   --   cea-aes128-gcm   |
   --   cea-aes192-gcm   |
   --   cea-aes256-gcm,
   --   ... -- Extensible
   --   }
```

```
   -- des-ede3-cbc and aes*-cbc are used with EnvelopedData and
   -- EncryptedData
   -- aes*-ccm are used with AuthEnvelopedData
   -- aes*-gcm are used with AuthEnvelopedData
```





```
   --
   -- Message Digest Algorithms
   --

   -- HMAC with SHA-224, HMAC with SHA-256, HMAC with SHA-384,
   -- HMAC with SHA-512 are specified in [HMAC-SHA2]

   -- Constrains the AuthenticatedData
   -- MessageAuthenticationCodeAlgorithm field
   -- Constrains the AuthEnvelopedData
   -- MessageAuthenticationCodeAlgorithm field

   MessageAuthenticationCodeAlgorithms MAC-ALGORITHM ::= {
     maca-sha1   |
     maca-sha224 |
     maca-sha256 |
     maca-sha384 |
     maca-sha512,
     ... -- Extensible
   }

   -- Would love to import the HMAC224-512 OIDS but they're not in a
   -- module (that I could find)

   maca-sha224 MAC-ALGORITHM ::= {
     IDENTIFIER id-hmacWithSHA224
     PARAMS TYPE NULL ARE preferredPresent
   }
```

```
   id-hmacWithSHA224 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 8 }

   maca-sha256 MAC-ALGORITHM ::= {
     IDENTIFIER id-hmacWithSHA256
     PARAMS TYPE NULL ARE preferredPresent
   }

   id-hmacWithSHA256 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 9 }
```





```
   maca-sha384 MAC-ALGORITHM ::= {
     IDENTIFIER id-hmacWithSHA384
     PARAMS TYPE NULL ARE preferredPresent
   }

   id-hmacWithSHA384 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 10 }

   maca-sha512 MAC-ALGORITHM ::= {
     IDENTIFIER id-hmacWithSHA512
     PARAMS TYPE NULL ARE preferredPresent
   }

   id-hmacWithSHA512 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549)
     digestAlgorithm(2) 11 }

   --
   -- Originator Public Key Algorithms
   --

   -- Constraints on KeyAgreeRecipientInfo OriginatorIdentifierOrKey
```

```
   -- OriginatorPublicKey algorithm field

   -- PARAMS are NULL

   OriginatorPKAlgorithms PUBLIC-KEY ::= {
     opka-ec,
     ... -- Extensible
   }

   opka-ec PUBLIC-KEY ::={
     IDENTIFIER id-ecPublicKey
     KEY ECPoint
     PARAMS TYPE CHOICE { n NULL, p ECPoint } ARE preferredAbsent
   }

   -- Format for both ephemeral and static public keys

   -- ECPoint ::= OCTET STRING
```





```
   -- Format of KeyAgreeRecipientInfo ukm field when used with
   -- ECMQV

   MQVuserKeyingMaterial ::= SEQUENCE {
     ephemeralPublicKey       OriginatorPublicKey,
     addedukm             [0] EXPLICIT UserKeyingMaterial OPTIONAL
   }

   -- 'SharedInfo' for input to KDF when using ECDH and ECMQV with
   -- EnvelopedData, AuthenticatedData, or AuthEnvelopedData

   ECC-CMS-SharedInfo ::= SEQUENCE {
     keyInfo         AlgorithmIdentifier { KeyWrapAlgorithm },
     entityUInfo [0] EXPLICIT OCTET STRING OPTIONAL,
     suppPubInfo [2] EXPLICIT OCTET STRING
   }

   --
```

```
-- S/MIME Capabilities
--

SMIME-CAPS ::= CLASS {
  &Type  OPTIONAL,
  &id    OBJECT IDENTIFIER UNIQUE
}
WITH SYNTAX {TYPE &Type IDENTIFIED BY &id }

SMIMECapability ::= SEQUENCE {
  capabilityID  SMIME-CAPS.&id({SMimeCapsSet}),
  parameters    SMIME-CAPS.
                  &Type({SMimeCapsSet}{@capabilityID}) OPTIONAL
}
```





```
SMimeCapsSet SMIME-CAPS ::= {
  cap-ecdsa-with-SHA1                  |
  cap-ecdsa-with-SHA224                |
  cap-ecdsa-with-SHA256                |
  cap-ecdsa-with-SHA384                |
  cap-ecdsa-with-SHA512                |
  cap-dhSinglePass-stdDH-sha1kdf       |
  cap-dhSinglePass-stdDH-sha224kdf     |
  cap-dhSinglePass-stdDH-sha256kdf     |
  cap-dhSinglePass-stdDH-sha384kdf     |
  cap-dhSinglePass-stdDH-sha512kdf     |
  cap-dhSinglePass-cofactorDH-sha1kdf  |
  cap-dhSinglePass-cofactorDH-sha224kdf |
```

```
    cap-dhSinglePass-cofactorDH-sha256kdf |
    cap-dhSinglePass-cofactorDH-sha384kdf |
    cap-dhSinglePass-cofactorDH-sha512kdf |
    cap-mqvSinglePass-sha1kdf             |
    cap-mqvSinglePass-sha224kdf           |
    cap-mqvSinglePass-sha256kdf           |
    cap-mqvSinglePass-sha384kdf           |
    cap-mqvSinglePass-sha512kdf,
    ... -- Extensible
  }

  --
  -- S/MIME Capabilities: ECDSA with SHA2 Algorithms
  --

  cap-ecdsa-with-SHA1 SMIME-CAPS ::= {
    TYPE NULL IDENTIFIED BY sa-ecdsaWithSHA1.&id }

  cap-ecdsa-with-SHA224 SMIME-CAPS ::= {
    TYPE NULL IDENTIFIED BY sa-ecdsaWithSHA224.&id }

  cap-ecdsa-with-SHA256 SMIME-CAPS ::= {
    TYPE NULL IDENTIFIED BY sa-ecdsaWithSHA256.&id }

  cap-ecdsa-with-SHA384 SMIME-CAPS ::= {
    TYPE NULL IDENTIFIED BY sa-ecdsaWithSHA384.&id }

  cap-ecdsa-with-SHA512 SMIME-CAPS ::= {
    TYPE NULL IDENTIFIED BY sa-ecdsaWithSHA512.&id }
```





```
  --
  -- S/MIME Capabilities: ECDH, Single Pass, Standard
  --

  cap-dhSinglePass-stdDH-sha1kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm IDENTIFIED BY dhSinglePass-stdDH-sha1kdf }

  cap-dhSinglePass-stdDH-sha224kdf SMIME-CAPS ::= {
```

```
    TYPE KeyWrapAlgorithm IDENTIFIED BY dhSinglePass-stdDH-sha224kdf }

  cap-dhSinglePass-stdDH-sha256kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm IDENTIFIED BY dhSinglePass-stdDH-sha256kdf }

  cap-dhSinglePass-stdDH-sha384kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm IDENTIFIED BY dhSinglePass-stdDH-sha384kdf }

  cap-dhSinglePass-stdDH-sha512kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm IDENTIFIED BY dhSinglePass-stdDH-sha512kdf }

  --
  -- S/MIME Capabilities: ECDH, Single Pass, Cofactor
  --

  cap-dhSinglePass-cofactorDH-sha1kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm
    IDENTIFIED BY dhSinglePass-cofactorDH-sha1kdf }

  cap-dhSinglePass-cofactorDH-sha224kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm
    IDENTIFIED BY dhSinglePass-cofactorDH-sha224kdf }

  cap-dhSinglePass-cofactorDH-sha256kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm
    IDENTIFIED BY dhSinglePass-cofactorDH-sha256kdf }

  cap-dhSinglePass-cofactorDH-sha384kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm
    IDENTIFIED BY dhSinglePass-cofactorDH-sha384kdf }

  cap-dhSinglePass-cofactorDH-sha512kdf SMIME-CAPS ::= {
    TYPE KeyWrapAlgorithm
    IDENTIFIED BY dhSinglePass-cofactorDH-sha512kdf }
```





```
  --
  -- S/MIME Capabilities: ECMQV, Single Pass, Standard
  --
```

```
cap-mqvSinglePass-sha1kdf SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm IDENTIFIED BY mqvSinglePass-sha1kdf }

cap-mqvSinglePass-sha224kdf SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm IDENTIFIED BY mqvSinglePass-sha224kdf }

cap-mqvSinglePass-sha256kdf SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm IDENTIFIED BY mqvSinglePass-sha256kdf }

cap-mqvSinglePass-sha384kdf SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm IDENTIFIED BY mqvSinglePass-sha384kdf }

cap-mqvSinglePass-sha512kdf SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm IDENTIFIED BY mqvSinglePass-sha512kdf }

END
```

Acknowledgements

The methods described in this document are based on work done by the ANSI X9F1 working group. The authors wish to extend their thanks to ANSI X9F1 for their assistance. The authors also wish to thank Peter de Rooij for his patient assistance. The technical comments of Francois Rousseau were valuable contributions.

Many thanks go out to the other authors of RFC 3278: Simon Blake-Wilson and Paul Lambert. Without the initial version of RFC3278 this version wouldn't exist.

The authors also wish to thank Alfred Hoenes, Paul Hoffman, Russ Housley, and Jim Schaad for their valuable input.

Author's Addresses

Sean Turner

IECA, Inc.
3057 Nutley Street, Suite 106
Fairfax, VA 22031
USA

Email: turners@ieca.com

Daniel R. L. Brown

Certicom Corp
5520 Explorer Drive #400
Mississauga, ON L4W 5L1
CANADA

Email: dbrown@certicom.com

## Full Copyright Statement



## Intellectual Property

ietf-ipr@ietf.org.

## Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).